8232
FLOATING POINT PROCESSOR

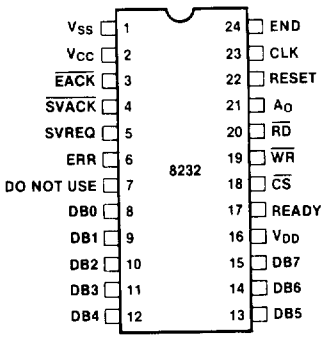
- **Compatible with Proposed IEEE Format and Existing Intel Floating Point Standard**
- **Single (32-Bit) and Double (64-Bit) Precision Capability**
- **Add, Subtract, Multiply and Divide Functions**
- **Stack Oriented Operand Storage**
- **General Purpose 8-Bit Data Bus Interface**
- **Standard 24-Pin Package**
- **12V and 5V Power Supplies**
- **Compatible with MCS-80™ and MCS-85™ Microprocessor Families**
- **Error Interrupt**
- **Direct Memory Access or Programmed I/O Data Transfers**
- **End of Execution Signal**
- **Advanced N-Channel Silicon Gate HMOS Technology**

The Intel® 8232 is a high performance floating-point processor unit (FPU). It provides single precision (32-bit) and double precision (64-bit) add, subtract, multiply and divide operations. It can be easily interfaced to enhance the computational capabilities of the host microprocessor.

The operand, result, status and command information transfers take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack by the host processor and a command is issued to perform an operation on the data stack. The results of the operation are available to the host processor from the stack.

Information transfers between the 8232 and the host processor can be handled by using programmed I/O or direct memory access techniques. After completing an operation, the 8232 activates an "end of execution" signal that can be used to interrupt the host processor.

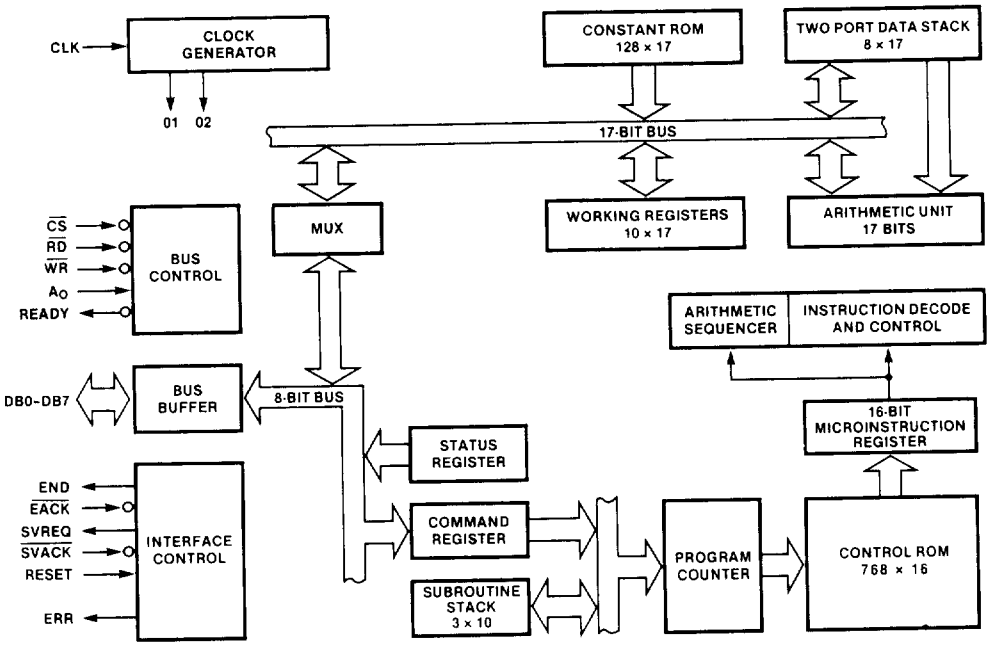
PIN CONFIGURATION



PIN NAMES

RESET	RESET
DB0-DB7	DATA BUS
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ DATA REGISTER
$\overline{WR}$	WRITE DATA OR COMMAND
A_0	COMMAND/DATA, INPUT
READY	READY, OUTPUT
END	END EXECUTION, OUTPUT
$\overline{EACK}$	END ACKNOWLEDGE, INPUT
SVREQ	SERVICE REQUEST, OUTPUT
$\overline{SVACK}$	SERVICE ACKNOWLEDGE, INPUT
CLK	CLOCK, INPUT
ERR	ERROR, OUTPUT

BLOCK DIAGRAM



SEPTEMBER 1979

PIN DESCRIPTION

Pin Name	Function
V_{CC}	+ 5V power supply
V_{DD}	+ 12V power supply
V_{SS}	Ground
CLK (Clock, Input)	An external timing source connected to the CLK input provides the necessary clocking.
RESET (Reset, Input)	A HIGH on this input causes initialization. Reset terminates any operation in progress, and clears the status register to zero. The internal stack pointer is initialized and the contents of the stack may be affected. After a reset the END output, the ERR output and the SVREQ output will be LOW. For proper initialization, RESET must be HIGH for at least five CLK periods following stable power supply voltages and stable clock.
A_0 (Command/ Data Select, Input	The A_0 input together with the $\overline{RD}$ and $\overline{WR}$ inputs determines the type of transfer to be performed on the data bus as follows:

A_0	$\overline{RD}$	$\overline{WR}$	Function
0	1	0	Push data byte into the stack
0	0	1	Pop data byte from the stack
1	1	0	Enter command
1	0	1	Read status
X	0	0	Undefined

END (End of Execution, Output)	A HIGH on this output indicates that execution of the current command is complete. This output will be cleared LOW by activating the $\overline{EACK}$ input LOW or performing any read or write operation or device initialization using the RESET. If $\overline{EACK}$ is tied LOW, the END output will be a pulse (see $\overline{EACK}$ description). Reading the status register while a command execution is in progress is allowed. However, any read or write operation clears the flip-flop that generates the END output. Thus, such continuous reading could conflict with internal logic setting of the END flip-flop at the end of command execution.
$\overline{EACK}$ (End Acknowledge, Input)	This input, when LOW, makes the END output go LOW. As mentioned earlier, HIGH on the END output signals completion of a command execution. The END signal is derived from an internal flip-flop which is clocked at the completion of a command. This flip-flop is clocked to the reset state when $\overline{EACK}$ is LOW. Consequently, if $\overline{EACK}$ is tied LOW, the END output will be a pulse that is approximately one CLK period wide.

Pin Name	Function
SVREQ (Service Request, Output)	A HIGH on this output indicates completion of a command. In this sense this output is the same as the END output. However, the SVREQ output will go HIGH at the completion of a command only when the Service Request Enable bit was set to 1. The SVREQ can be cleared (i.e., go LOW) by activating the $\overline{SVACK}$ input LOW or initializing the device using the RESET. Also, the SVREQ will be automatically cleared after completion of any command that has the service request bit as 0.
$\overline{SVACK}$ (Service Acknowledge, Input)	A LOW on this input clears SVREQ. If the $\overline{SVACK}$ input is permanently tied LOW, it will conflict with the internal setting of the SVREQ output. Thus, the SVREQ indication cannot be relied upon if the $\overline{SVACK}$ is tied LOW.
DB0-DB7 (Data Bus, Input/Output)	These eight bidirectional lines are used to transfer command, status and operand information between the device and the host processor. DB0 is the least significant and DB7 is the most significant bit position. HIGH on a data bus line corresponds to 1 and LOW corresponds to 0. When pushing operands on the stack using the data bus, the least significant byte must be pushed first and the most significant byte last. When popping the stack to read the result of an operation, the most significant byte will be available on the data bus first and the least significant byte will be the last. Moreover, for pushing operands and popping results, the number of transactions must be equal to the proper number of bytes appropriate for the chosen format. Otherwise, the internal byte pointer will not be aligned properly. The single precision format requires 4 bytes and double precision format requires 8 bytes.
ERR (Error, Output)	This output goes HIGH to indicate that the current command execution resulted in an error condition. The error conditions are: attempt to divide by zero, exponent overflow and exponent underflow. The ERR output is cleared LOW on read status register operation or upon RESET. The ERR output is derived from the error bits in the status register. These error bits will be updated internally at an appropriate time during a command execution. Thus, ERR output going HIGH may not coincide with the completion of a command. Reading of the status register can be performed while a command execution is in progress. However, it should be noted that reading the status register

Pin Name	Function
	clears the ERR output. Thus, reading the status register while a command execution is in progress may result in an internal conflict with the ERR output.
$\overline{CS}$ (Chip Select, Input)	This input must be LOW to accomplish any read or write operation to the 8232. To perform a write operation, appropriate data is presented on DB0 through DB7 lines, appropriate logic level on the A_0 input and the $\overline{CS}$ input is made LOW. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH and $\overline{CS}$ is LOW, READY goes LOW. However, actual writing into the 8232 cannot start until $\overline{WR}$ is made LOW. After initiating the write operation by the HIGH to LOW transition on the $\overline{WR}$ input, the READY output will go HIGH, indicating the write operation has been acknowledged. The $\overline{WR}$ input can go HIGH after READY goes HIGH. The data lines, A_0 input and the $\overline{CS}$ input can change when appropriate hold time requirements are satisfied. See write timing diagram for details. To perform a read operation an appropriate logic level is established on the A_0 input and $\overline{CS}$ is made LOW. The READY output goes LOW because $\overline{WR}$ and $\overline{RD}$ inputs are HIGH. The read operation does not start until the $\overline{RD}$ input goes LOW. READY will go HIGH indicating that read operation is complete and the required information is available on the DB0 through DB7 lines. This information will remain on the data lines as long as $\overline{RD}$ is LOW. The $\overline{RD}$ input can return HIGH anytime after READY goes HIGH. The $\overline{CS}$ input and A_0 input can change anytime after $\overline{RD}$ returns HIGH. See read timing diagram for details. If the $\overline{CS}$ is tied LOW permanently, READY will remain LOW until the next 8232 read or write access.
$\overline{RD}$ (Read, Input)	A LOW on this input is used to read information from an internal location and gate that information onto the data bus. The $\overline{CS}$ input must be LOW to accomplish the read operation. The A_0 input determines what internal location is of interest. See A_0, $\overline{CS}$ input descriptions and read timing diagram for details. If the END output was HIGH, performing any read operation will make the END output go LOW after the HIGH to LOW transition of the $\overline{RD}$ input (assuming $\overline{CS}$ is LOW). If the ERR output was HIGH, performing a status register read operation will make the ERR output LOW. This will happen after the HIGH to LOW transition of the $\overline{RD}$ input (assuming $\overline{CS}$ is LOW).

Pin Name	Function
$\overline{WR}$ (Write, Input)	A LOW on this input is used to transfer information from the data bus into an internal location. The $\overline{CS}$ must be LOW to accomplish the write operation. The A_0 determines which internal location is to be written. See A_0, $\overline{CS}$ input descriptions and write timing diagram for details. If the END output was HIGH, performing any write operation will make the END output go LOW after the LOW to HIGH transition of the $\overline{WR}$ input (assuming $\overline{CS}$ is LOW).
READY (Ready, Output)	This output is a handshake signal used while performing read or write transactions with the 8232. If the $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH, the READY output goes LOW with the $\overline{CS}$ input in anticipation of a transaction. If $\overline{WR}$ goes LOW to initiate a write transaction with proper signals established on the DB0–DB7, A_0 inputs, the READY will return HIGH indicating that the write operation has been accomplished. The $\overline{WR}$ can be made HIGH after this event. On the other hand, if a read operation is desired, the $\overline{RD}$ input is made LOW after activating $\overline{CS}$ LOW and establishing proper A_0 input. (The READY will go LOW in response to $\overline{CS}$ going LOW.) The READY will return HIGH, indicating completion of read. The $\overline{RD}$ can return HIGH after this event. It should be noted that a read or write operation can be initiated without any regard to whether a command execution is in progress or not. Proper device operation is assured by obeying the READY output indication as described.

8232 FUNCTIONAL DESCRIPTION

Major functional units of the 8232 are shown in the block diagram. The 8232 employs a microprogram controlled stack oriented architecture with 17-bit wide data paths.

The Arithmetic Unit receives one of its operands from the Operand Stack. This stack is an eight word by 17-bit two port memory with last in-first out (LIFO) attributes. The second operand to the Arithmetic Unit is supplied by the internal 17-bit bus. In addition to supplying the second operand, this bidirectional bus also carries the results from the output of the Arithmetic Unit when required. Writing into the Operand Stack takes place from this internal 17-bit bus when required. Also connected to this bus are the Constant ROM and Working Registers. The ROM provides the required constants to perform the mathematical operations while the Working Registers provide storage for the intermediate values during command execution.

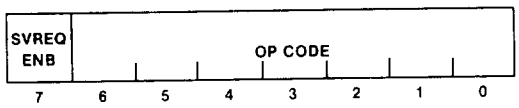
Communication between the external world and the 8232 takes place on eight bidirectional input/output lines, DB0 through DB7 (Data Bus). These signals are gated to the internal 8-bit bus through appropriate interface and buffer circuitry. Multiplexing facilities exist for bidirectional communication between the internal eight and 17-bit buses. The Status Register and Command Register are also located on the 8-bit bus.

The 8232 operations are controlled by the microprogram contained in the Control ROM. The Program Counter supplies the microprogram addresses and can be partially loaded from the Command Register. Associated with the Program Counter is the Subroutine Stack where return addresses are held during subroutine calls in the microprogram. The Microinstruction Register holds the current microinstruction being executed. The register facilitates pipelined microprogram execution. The Instruction Decode logic generates various internal control signals needed for the 8232 operation.

The Internal Control logic receives several external inputs and provides handshake related outputs to facilitate interfacing the 8232 to microprocessors.

COMMAND FORMAT

The operation of the 8232 is controlled from the host processor by issuing instructions called commands. The command format is shown below.



The command consists of 8 bits; the least significant 7 bits specify the operation to be performed as detailed in Table 1. The most significant bit is the Service Request Enable bit. This bit must be a 1 if SVREQ is to go HIGH at the end of executing a command.

The commands fall into three categories: single precision arithmetic, double precision arithmetic and data manipulation. There are four arithmetic operations that can be performed with single precision (32-bit) or double precision (64-bit) floating-point numbers: add, subtract, multiply and divide. These operations require two operands. The 8232 assumes that these operands are located in the internal stack as Top of Stack (TOS) and Next on Stack (NOS). The result will always be returned to the previous NOS which becomes the new TOS. Results from an operation are of the same precision and format as the operands. The results will be rounded to preserve the accuracy. The actual data formats and rounding procedures are described in a later section. In addition to the arithmetic operations, the 8232 implements eight data manipulating operations. These include changing the sign of a double or single precision operand located in TOS, exchanging single precision operands located at TOS and NOS, as well as copying and popping single or double precision operands. See also the sections on status register and operand formats.

The execution times of the commands are all data dependent. Table 2 shows one example of each command execution time.

OPERAND ENTRY

The 8232 commands operate on the operands located at the TOS and NOS. Results are returned to the stack at NOS and then popped to TOS. The operands required for the 8232 are one of two formats — single precision floating-point (4 bytes) or double precision floating-point (8 bytes). The result of an operation has the same format as the operands. In other words, operations using single precision quantities always result in a single precision result, while operations involving double precision quantities will result in double precision result.

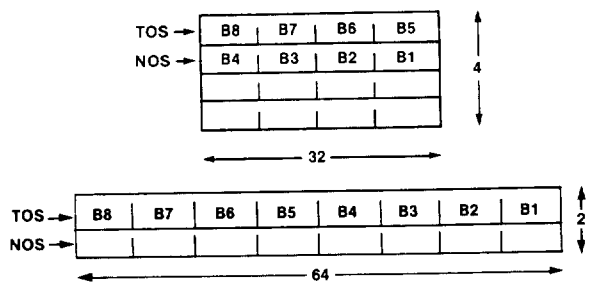
Operands are always entered into the stack least significant byte first and most significant byte last. The following procedure must be followed to enter operands into the stack:

1. The lower significant operand byte is established on the DB0-DB7 lines.
2. A LOW is established on the A_0 input to specify that data is to be entered into the stack.
3. The $\overline{CS}$ input is made LOW. Whenever the $\overline{WR}$ and $\overline{RD}$ inputs are HIGH, the READY output will follow the $\overline{CS}$ input. Thus, READY output will become LOW.
4. After appropriate set up time (see timing diagrams), the $\overline{WR}$ input is made LOW.
5. Sometime after this event, READY will return HIGH to indicate that the write operation has been acknowledged.
6. Any time after the READY output goes HIGH, the $\overline{WR}$ input can be made HIGH. The DB0-DB7, A_0 and $\overline{CS}$ inputs can change after appropriate hold time requirements are satisfied (see timing diagrams).

The above procedure must be repeated until all bytes of the operand are pushed into the stack. It should be noted that for single precision operands 4 bytes should be pushed and 8 bytes must be pushed for double precision. Not pushing all the bytes of a quantity will result in byte pointer misalignment.

The 8232 stack can accommodate four single precision quantities or two double precision quantities. Pushing more quantities than the capacity of the stack will result in loss of data which is usual with any LIFO stack.

The stack can be visualized as shown below:



COMMAND INITIATION

After properly positioning the required operands in the stack, a command may be issued. The procedure for initiating a command execution is the same as that described above for operand entry, except that the A_0 input is HIGH.

Commands Bits								Mnemonic	Description
7	6	5	4	3	2	1	0		
X	0	0	0	0	0	0	1	SADD	Add TOS to NOS single precision and result to NOS. Pop stack.
X	0	0	0	0	0	1	0	SSUB	Subtract TOS from NOS single precision and result to NOS. Pop stack.
X	0	0	0	0	0	1	1	SMUL	Multiply NOS by TOS single precision and result to NOS. Pop stack.
X	0	0	0	0	1	0	0	SDIV	Divide NOS by TOS single precision and result to NOS. Pop stack.
X	0	0	0	0	1	0	1	CHSS	Change sign of TOS single precision operand.
X	0	0	0	0	1	1	0	PTOS	Push single precision operand on TOS to NOS.
X	0	0	0	0	1	1	1	POPS	Pop single precision operand from TOS. NOS becomes TOS.
X	0	0	0	1	0	0	0	XCHS	Exchange TOS with NOS single precision.
X	0	1	0	1	1	0	1	CHSD	Change sign of TOS double precision operand.
X	0	1	0	1	1	1	0	PTOD	Push double precision operand on TOS to NOS.
X	0	1	0	1	1	1	1	POPD	Pop double precision operand from TOS. NOS becomes TOS.
X	0	0	0	0	0	0	0	CLR	CLR status.
X	0	1	0	1	0	0	1	DADD	Add TOS to NOS double precision and result to NOS. Pop stack.
X	0	1	0	1	0	1	0	DSUB	Subtract TOS from NOS double precision and result to NOS. Pop stack.
X	0	1	0	1	0	1	1	DMUL	Multiply NOS by TOS double precision and result to NOS. Pop stack.
X	0	1	0	1	1	0	0	DDIV	Divide NOS by TOS double precision and result to NOS. Pop stack.

Notes: X = Don't care. Operation for bit combinations not listed above is undefined.

Table 1. Command Decoding Table

Command	TOS	NOS	Result	Clock Periods
SADD	3F800000	3F800000	40000000	58
SSUB	3F800000	3F800000	00000000	56
SMUL	40400000	3FC00000	40900000	198
SDIV	3F800000	40000000	3F000000	228
CHSS	3F800000	—	BF800000	10
PTOS	3F800000	—	—	16
POPS	3F800000	—	—	14
XCHS	3F800000	40000000	—	26
CHSD	3FF0000000000000	—	BFF0000000000000	24
PTOD	3FF0000000000000	—	—	40
POPD	3FF0000000000000	—	—	26
CLR	3FF0000000000000	—	—	4
DADD	3FF000000A000000	800000000000000	3FF00000A0000000	578
DSUB	3FF00000A0000000	800000000000000	3FF00000A0000000	578
DMUL	BFF800000000000	3FF800000000000	C002000000000000	1748
DDIV	BFF800000000000	3FF800000000000	BFF0000000000000	4560

Note: TOS, NOS and result are in hexadecimal; clock period is in decimal.

Table 2. Execution Times

An attempt to issue a new command while the current command execution is in progress is allowed. Under these circumstances, the READY output will not go HIGH until the current command execution is completed.

REMOVING THE RESULTS

Result from an operation will be available at the TOS. Results can be transferred from the stack to the data bus by reading the stack.

When the stack is read for results, the most significant byte is available first and the least significant byte last.

A result is always of the same precision as the operands that produced it. Thus, when the result is taken from the stack, the total number of bytes popped out should be appropriate with the precision — single precision results are 4 bytes and double precision results are 8 bytes. The following procedure must be used for reading the result from the stack:

1. A LOW is established on the A_0 input.
2. The $\overline{CS}$ input is made LOW. When $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH, the READY output follows the $\overline{CS}$ input, thus READY will be LOW.

3. After appropriate set up time (see timing diagrams), the $\overline{RD}$ input is made LOW.
4. Sometime after this, READY will return HIGH, indicating that the data is available on the DB0-DB7 lines. This data will remain on the DB0-DB7 lines as long as the $\overline{RD}$ input remains LOW.
5. Any time after READY goes HIGH, the $\overline{RD}$ input can return HIGH to complete the transaction.
6. The $\overline{CS}$ and A_0 inputs can change after appropriate hold time requirements are satisfied (see timing diagram).
7. Repeat this procedure until all bytes appropriate for the precision of the result are popped out.

Reading of the stack does not alter its data; it only adjusts the byte pointer. If more data is popped than the capacity of the stack, the internal byte pointer will wrap around and older data will be read again, consistent with the LIFO stack.

READING STATUS REGISTER

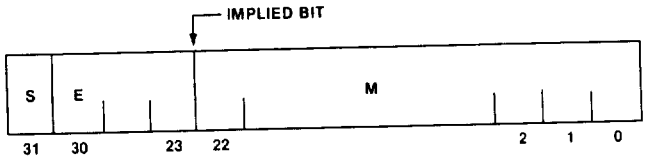
The 8232 status register can be read without any regard to whether a command is in progres or not. The only implication that has to be considered is the effect this might have on the END and ERR outputs discussed in the signal descriptions.

The following procedure must be followed to accomplish status register reading:

1. Establish HIGH on the A_0 input.
2. Establish LOW on the $\overline{CS}$ input. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are HIGH, READY will follow the $\overline{CS}$ input. Thus, READY will go LOW.
3. After appropriate set up time (see timing diagram), $\overline{RD}$ is made LOW.
4. Sometime after the HIGH to LOW transition of $\overline{RD}$, READY will become HIGH, indicating that status register contents are available on the DB0-DB7 lines. These lines will contain this information as long as $\overline{RD}$ is LOW.
5. The $\overline{RD}$ input can be returned HIGH any time after READY goes HIGH.
6. The A_0 input and $\overline{CS}$ input can change after satisfying appropriate hold time requirements (see timing diagram).

DATA FORMATS

The 8232 handles floating-point quantities in two different formats — single precision and double precision. The single precision quantities are 32-bits long, as shown below:



Bit 31:

S = Sign of the mantissa. One represents negative and 0 represents positive.

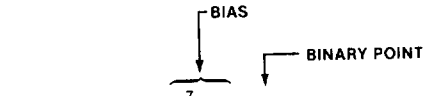
Bits 23-30:

E = These 8 bits represent a biased exponent. The bias is $2^7 - 1 = 127$.

Bits 0-22:

M = 23-bit mantissa. Together with the sign bit, the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit (bit 22) of the mantissa. In other words, the mantissa is assumed to be a 24-bit normalized quantity and the most significant bit, which will always be a 1 due to normalization, is implied. The 8232 restores this implied bit internally before performing arithmetic, normalizes the result and strips the implied bit before returning the results to the external data bus. The binary point is between the implied bit and bit 22 of the mantissa.

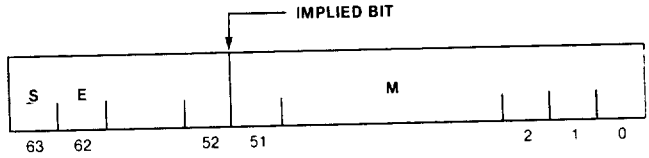
The quantity N represented by the above notation is



$$N = (-1)^S \, 2^{E-(2^7-1)} \, (1.M)$$

Provided E≠0 (reserved for 0) or all 1's (illegal).

A double precision quantity consists of the mantissa sign bit, an 11-bit biased exponent (E), and a 52-bit mantissa (M). The bias for double precision quantities is $2^{10} - 1$. The double precision format is illustrated below.



Bit 63:

S = Sign of the mantissa. One represents negative and 0 represents positive.

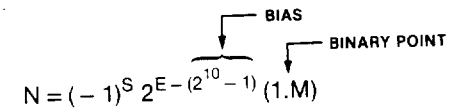
Bits 52-62:

E = These 11 bits represent a biased exponent. The bias is $2^{10} - 1 = 1023$.

Bits 0-51:

M = 52-bit mantissa. Together with the sign bit the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit (bit 51) of the mantissa. In other words, the mantissa is assumed to be a 53-bit normalized quantity and the most significant bit, which will always be a 1 due to normalization, is implied. The 8232 restores this implied bit internally before performing arithmetic, normalizes the result and strips the implied bit before returning the result to the external data bus. The binary point is between the implied bit and bit 51 of the mantissa.

The quantity N represented by the above notation is



$$N = (-1)^S \, 2^{E-(2^{10}-1)} \, (1.M)$$

Provided E≠0 (reserved for 0) or all 1's (illegal).